

Grammar Engineering for CCG using Ant and XSLT*

Scott Martin, Rajakrishnan Rajkumar, and Michael White

Ohio State University

Department of Linguistics

{scott,raja,mwhite}@ling.ohio-state.edu

Overview

Corpus conversion and grammar extraction have traditionally been portrayed as tasks that are performed once and never again revisited (Burke et al., 2004). We report the successful implementation of an approach to these tasks that facilitates the improvement of grammar engineering as an evolving process. Taking the standard version of the CCGbank (Hockenmaier and Steedman, 2007) as input, our system then introduces greater depth of linguistic insight by augmenting it with attributes the original corpus lacks: Propbank roles and head lexicalization for case-marking prepositions (Boxwell and White, 2008), derivational re-structuring for punctuation analysis (White and Rajkumar, 2008), named entity annotation and lemmatization. Our implementation applies successive XSLT transforms controlled by Apache Ant (<http://ant.apache.org/>) to an XML translation of this corpus, finally producing an OpenCCG grammar (<http://openccg.sourceforge.net/>). This design is beneficial to grammar engineering both because of XSLT's unique suitability to performing arbitrary transformations of XML trees and the fine-grained control that Ant provides. The resulting system enables state-of-the-art BLEU scores for surface realization on section 23 of the CCGbank.

1 Design

Rather than transforming the corpus, it would be simple to introduce several of the corpus aug-

mentations that we make (e.g. punctuation restructuring) during grammar extraction. However, machine learning applications (e.g., realization ranking) benefit when the corpus and extracted grammar are consistent. A case in point: annotating the corpus with named entities, then using n-gram models with words replaced by their class labels to score realization.

Accordingly, our pipeline design starts by generating an XML version of the CCGbank using JavaCC (<http://javacc.dev.java.net/>) from the original corpus. Next, conversion and extraction transforms are applied to create a converted corpus (also in XML) and extracted grammar (in OpenCCG format).

We refactored our original design to separate the grammar engineering task into several configurable processes using Ant tasks. This simplifies process management, speeds experiment iterations, and facilitates the comparison of different grammar engineering strategies.

2 Implementation

It seemed natural to implement our pipeline procedure in XSLT since both OpenCCG grammars and our CCGbank translation are represented in XML. Aside from its inherent attributes, XSLT requires no re-compilation as a result of being an interpreted language. Also, because both conversion and extraction use a series of transforms in a chain, each required sub-step can be split into as many XSLT transforms as desired.

Both the conversion and extraction steps were implemented by extending Ant with custom tasks as configuring Ant tasks requires no

*This work was supported in part by NSF grant no. IIS-0812297.

source editing or compilation. Ant is particularly well-suited to this process because, like OpenCCG (whose libraries are used in the extraction phase), it is written in Java. Our system also employs the Ant-provided `javacc` task, invoking the JavaCC parser to translate the CCGbank to XML. This approach is preferable to a direct Java implementation because it keeps source code and configuration separate, allowing for more rapid grammar engineering iterations.

Our particular implementation harnesses Ant’s built-in `FileSet` (for specification of groups of corpus files) and `FileList` (for reuse of series of XSLT transforms) data types. The first of our extension tasks, `convert`, encapsulates the conversion process while the second task, `extract`, implements the grammar extraction procedure for a previously-converted corpus.

3 Experimental Impact

Our conversion process currently supports various experiments by including only specified transforms. We gain the ability to create corpora with various combinations of attributes, among them punctuation annotation, semantic class information, and named entities (lack of space precludes inclusion of examples here; see <http://www.ling.ohio-state.edu/~scott/publications/grammareng/>). In addition to extracting grammars, the extraction task employs a constrained parser to create logical forms (LFs) for surface realization and extracts SRILM training data for realization scoring. This task also enables feature extraction from LF graphs for training during supertagging for realization (Espinosa et al., 2008).

Our design supports comprehensive experimentation and has helped facilitate recent efforts to investigate factors impacting surface realization, such as semantic classes and named entities. Our initial results reported in (White et al., 2007) record 69.7% single-rooted LFs with a BLEU score of 0.5768. But current figures stand at 95.8% single-rooted LFs and a state-of-the-art BLEU score of 0.8506 on section 23 of the CCGbank. (Fragmentary LFs result when at

least one semantic dependency is missing from the LF graph.) In achieving these results, improvements in the grammar engineering process have been at least as important as improvements in the statistical models.

4 Conclusions and Future Work

We designed and implemented a system that facilitates the process of grammar engineering by separating conversion and extraction steps into a pipeline of XSLT transforms. Our Ant implementation is highly configurable and has positive effects on our grammar engineering efforts, including increased process control and a shortened testing cycle for different grammar engineering approaches. Future work will focus on increasing the number of single-rooted LFs and integrating this system with OpenCCG.

References

- [Boxwell and White2008] Stephen Boxwell and Michael White. 2008. Projecting Propbank roles onto the CCGbank. In *Proc. LREC-08*.
- [Burke et al.2004] Michael Burke, Aoife Cahill, Mairead Mccarthy, Ruth O’Donovan, Josef Genabith, and Andy Way. 2004. Evaluating automatic LFG F-structure annotation for the Penn-II treebank. *Research on Language and Computation*, 2:523–547, December.
- [Espinosa et al.2008] Dominic Espinosa, Michael White, and Dennis Mehay. 2008. Hypertagging: Supertagging for surface realization with CCG. In *Proc. ACL-08: HLT*.
- [Hockenmaier and Steedman2007] Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.
- [White and Rajkumar2008] Michael White and Rajakrishnan Rajkumar. 2008. A more precise analysis of punctuation for broad-coverage surface realization with CCG. In *Proc. of the Workshop on Grammar Engineering Across Frameworks (GEAF08)*.
- [White et al.2007] Michael White, Rajakrishnan Rajkumar, and Scott Martin. 2007. Towards broad coverage surface realization with CCG. In *Proc. of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UC-NLG+MT)*.